Safety Cases for Advanced Control Software: Safety Case Patterns

Robert Alexander, Tim Kelly, Zeshan Kurd, John McDermid

Department of Computer Science
University of York

15th October 2007

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
04-01-2008	Final Report	20 June 2007 - 15-Sep-08

4. TITLE AND SUBTITLE

Safety Cases for Advanced Control Software

5a. CONTRACT NUMBER
FA8655-07-1-3025

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Professor John A McDermid

5d. PROJECT NUMBER

5d. TASK NUMBER

5e. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of York
Heslington
York YO10 5DD
United Kingdom

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
Unit 4515 BOX 14
APO AE 09421

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Grant 07-3025

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report results from a contract tasking University of York as follows: The project will undertake one activity: Produce a unified (generic) approach to developing safety cases for adaptive avionics and software and identifying a 'way ahead' to develop and validate the approach, based on the outline produced in the preceding NASA project.

15. SUBJECT TERMS
EOARD

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18, NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			JAMES LAWTON Ph. D.
UNCLAS	UNCLAS	UNCLAS	UL	29	**19b. TELEPHONE NUMBER** *(Include area code)* +44 (0)1895 616187

1. Introduction

As with the previous report [1], our focus in addressing safety cases for 'advanced' control systems is to concentrate on the class of *adaptive* systems. A system can be considered adaptive if its behaviour cannot be predicted solely from knowledge of its initial software design and state. The behaviour of an adaptive system is the product of its initial state and the adaptations (state changes) that have taken place according to the stimuli it has encountered.

Adaptive systems can be introduced to improve safety (e.g. to continue to control an aircraft safely in the event of losing a control surface) or to improve other system characteristics (e.g. to improve the fuel consumption of an aero-engine). The motivation for introducing an adaptive capability has a significant impact on the nature of the required safety argument. Where improved safety is the *goal* of the adaptation, the safety argument must justify that the adaptive system is capable of reducing some of the risks associated with hazards already present with the equipment under control. At the same, it is necessary to ensure that the introduction of the adaptive capability does not introduce new, or increase existing, risks. Where adaptation is being introduced for reasons other than safety, safety can be viewed as a *constraint*. The principal concern is that the adaptive capability doesn't introduce new, or increase existing, risks.

The novelty and perceived unpredictability of adaptive systems can make safety engineers and regulators look sceptically upon their potential use in safety-critical applications. (Indeed, the international safety standard IEC 61508 [2] advises against the use of artificial intelligence techniques for the highest integrity applications.) This increases the need to establish compelling safety cases that assure their safe use in safety-critical applications. A number of arguments will be required as part of the safety case for adaptive systems. For example, it may be necessary to argue that levels of safety experienced with conventional (non-adaptive) systems are maintained, that an acceptable balance has been made between risk-reduction and cost, and that an adaptation mechanism is inherently safe. This report presents the patterns of argument that can be used in structuring a safety case for an adaptive system.

The structure of this report is as follows. Section 2 provides an overview of existing (software) safety standards and discusses the extent to which compliance with these standards will result in a compelling case for safety for adaptive systems. Section 3 presents a collection of fourteen argument patterns – expressed using the Goal Structuring Notation (GSN) – that, in composition, can be used to establish the principal arguments of safety required for an adaptive system. Section 4 provides a summary of the approach outlined.

2. The Role of Existing (Software) Safety Standards

A large number of software safety standards exist to define required software safety assurance practice. These standards vary in their requirements. Whilst some of this variance is in the detail (e.g. favoured verification methods), some large differences in philosophy remain. One such philosophical difference is between so-called *process assurance-based* safety standards – such as DO-178B [3] – and *product evidence-based* safety standards – such as UK Defence Standard 00-56 Issue 4 [4]. In this section we will discuss whether compliance with these standards will establish a compelling safety case for an adaptive system.

2.1 Process Assurance-Based Certification

A number of software assurance standards – such as DO-178B [3] and IEC 61508 [2] – are described as being "process-based", in that they define a set of practices to be adhered to in the development, verification and validation of software. In such standards the software processes are typically prescribed according to the criticality of software failure. In the civil aerospace domain Development Assurance Levels (DALs) (e.g. see [3]) are used to define the level of rigour required. In the European rail, process industry, and automotive domains Safety Integrity Levels (SILs) (e.g. see [2]) are used. SILs and DALs are similar concepts, but differ in the details of their allocation, requirements and application.

Both SILs and DALs define the level of risk reduction expected from a software system. The greater the criticality of a software-involved system, the greater the risk reduction is necessarily attributed to that system. SILs and DALs can also be thought of as specifying the required degree of freedom of the system from flaw. For software systems, this particularly relates to the degree of freedom from systematic errors in the design – introduced through failings in the software production process. Processes and techniques are specified for each SIL / DAL. The higher the SIL / DAL, the more demanding are the requirements on the software production process.

The following quote from the introduction to DO-178B summarises the philosophy behind the organisation of the standard:

> *"These guidelines are in the form of:*
> *- Objectives for software life cycle processes.*
> *- Descriptions of activities and design considerations for achieving those objectives.*
> *- Descriptions of the evidence that indicates that the objectives have been satisfied."*

DO-178B defines an outline software life-cycle (as shown in Figure 1). The main stages of this life-cycle are:

- Software Requirements (both High-Level and Low-Level)
- Software Design
- Software Coding
- Integration

Many of the requirements of DO-178B are expressed over this model of the process. For example, there are requirements (called 'objectives' in DO-178B) concerned with the consistency of the artefacts produced at each stage, and of the compliance between the artefacts of one stage (e.g. source code) and the artefacts of another (e.g. low-level requirements). DO-178B places a strong emphasis on traceability, and the human review of artefacts (such as requirements). DO-178B also strongly favours testing as the primary means of verification.

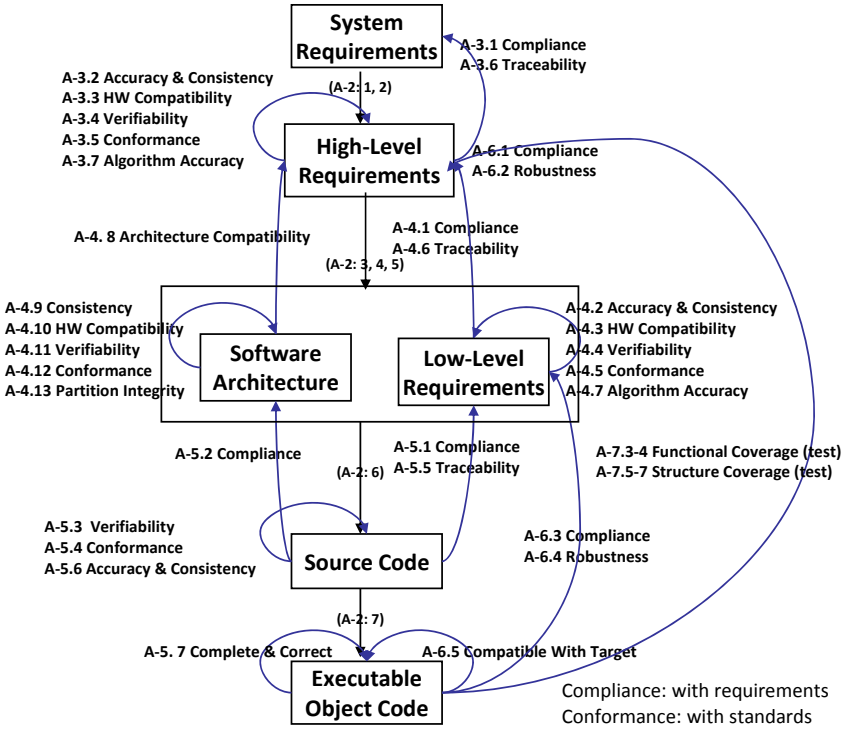

Figure 1 – DO-178B Life-cycle and Objectives

The objectives of DO-178B vary according to DAL. At level D, the lowest level, 28 objectives are defined (covering aspects such as configuration management, tool qualification, and high-level requirements coverage). At level C, a further 29 objectives are added (covering aspects such as statement coverage, and testing of low-level requirements). At level B, a further 8 objectives are added (covering aspects such as decision coverage). Finally, at level A the requirement for MC/DC coverage is added, together with greater source code to object code traceability. In total, 66 objectives are defined for a Level A compliant software development.

2.1.1 Applying Process-Assurance Approaches to Adaptive Systems Justification

It is possible to applying a process-assurance approach to the development of an adaptive system. However, it is important to recognise a key difference that exists between the development of conventional software system and the development of an adaptive software system. In the development of conventional systems, requirements are refined, decomposed and allocated to the point where a deterministic solution is produced. In the development of adaptive systems, the ultimate refinement of operational behaviour is performed at run-time. Development of an adaptive system is concerned with the creation of a system that can, in operation, alter its behaviour response to adaptation stimuli. Therefore, assurance of the development process of an adaptive system may ultimately fail to provide a compelling basis for an argument of acceptably safe behaviour. Where adaptation following the initial implementation of an adaptive system can have an impact on safety this must be included with the scope of the safety argument.

It is hard to establish a safety case for an adaptive system purely from the satisfaction of positively expressed requirements. Such requirements describe the desired behaviour of the adaptation *mechanism*. Safety requirements often have a negative focus [5]. They define properties that the software should not exhibit *in operation*. This can create difficulties in assessing the completeness of any testing performed purely from a perspective of implementation 'compliance'.

The lifecycle of software assurance standards such as DO-178B can also pose a problem for the development of adaptive systems. The lifecycle shown in Figure 1 assumes the 'conventional' progression of requirements to design to implementation. The development of adaptive systems does not always fit neatly within such a process. For example, adaptive systems are often employed in situations where an exact specification of required behaviour cannot be provided. System development may start from an intentionally incomplete specification that will need to be refined through an initial period of training. The development of safety requirements will have to be determined incrementally as the behaviour of the adaptive system emerges through adaptation. In this regard, the lifecycle of adaptive system development may be much closer to evolutionary software development lifecycles such as Boehm's spiral model [6].

A limitation of DO-178B is that it currently strongly emphasises testing as the primary means of software verification. The results of *exhaustive* testing provide strong evidence for claims about software behaviour. However, it is unlikely to be possible to provide adequate test coverage of an adaptive system. The behaviour of the system depends not only on its static structure and immediate stimuli but on the behaviour that it has learned, which in turn depends on the adaptation stimuli (e.g. training data) that it has learnt from. In the case of online learning, the behaviour of the system depends on the current stimuli and (potentially) all previous stimuli. Testing is likely to be impractical, particularly if the input space is large [7].

There are, in any case, problems with the use of testing for strong safety claims about conventional software. For example, Littlewood and Strigini note in [8] that, at best, statistical testing can show a failure rate of around 10^{-4} per operating hour. This figure is inadequate for many safety-critical software applications.

In the revision of DO-178B (to produce DO-178C) proposals have been made to generalise the wording to call for 'verification' instead of specific methods (such as 'testing' or 'review'). This approach "opens the door" for alternative forms of evidence to be selected without needing to be justified as a deviation from the defined verification approach. The use of alternative (analytic) forms of evidence is discussed further in section 2.2.1.

2.2 Product-Based Certification

Product-based certification focuses on the construction of well-structured and reasoned safety arguments. Arguments are required to demonstrate the satisfaction of product-specific safety objectives derived from hazard analysis; justify the acceptability of safety, based upon product-specific and targeted evidence; and (potentially) justify the determination of the safety objectives and selection of evidence. The arguments and evidence required to justify acceptable safety form the safety case, and are often summarised in a safety case report [9]. This is the approach required by the UK Defence Standard 00-56 Issue 4 [4].

For software, product-based certification demands that software level safety claims are hazard-based – i.e. they concern failures of the software that are believed to lead to system level hazards. Ideally, these claims should be derived from a system-level safety case. In a system-level safety case a claim relating to a specific behaviour of the software may be seen as a contributing, but undeveloped argument. From the perspective of software safety, such claims are the starting point for the construction of the software safety case. This is the intended relationship between system and software level safety cases under UK Defence Standard 00-56 Issue 4 [4]. This means that the focus in the software level safety arguments is on "demonstrating the safety of …", rather than "demonstrating the development of …" the software system. Arguments and evidence about the development process followed are not of interest unless they can somehow be specifically related to the product-specific software safety claims.

The principle that the risks should be reduced As Low As Reasonably Practicable (ALARP) is core to the UK Defence Standard 00-56. The ALARP principle (discussed in more detail later in section 3.7) allows the benefit associated with the risks being posed, and the proportionality of the costs involved in risk reduction, to be included in the arguments of risk acceptance. As discussed later, the ALARP principle is potentially valuable when attempting to argue the acceptability of risks associated with adaptive systems – particularly in situations where an adaptive capability is brought in for reasons other than safety. The ALARP principle is now also included in US MilStd 882 (at draft E) [10] as a means of justifying risk acceptance.

Safety Integrity Levels (SILs) no longer form part of the requirements of UK Defence Standard 00-56. Instead, 00-56 requires that the level of evidence presented in the safety case ought to be chosen according to the level of risk associated with the system:

> *"The quantity and quality of the evidence shall be commensurate with the potential risk posed by the system and the complexity of the system."*

This requirement is coupled with the following guidance on the relative strength of different forms of argument and evidence:

> *"In general, arguments based on explicit, objective evidence are more compelling than those that appeal to judgement or custom and practice. It is therefore recommended that any argument should be developed in accordance with the following precedence:*
>
> - *Deductive, where the conclusion is implicit in the evidence used to support the argument.*
> - *Inductive, where the argument is firmly based on the evidence presented, but extrapolates beyond the available evidence.*
> - *Judgmental, where expert testimony, or appeal to custom and practice is necessary to support the conclusion."*

By comparison, US MilStd 882E has, in guidance, the concept of *Software Control Categories* as a means of expressing the criticality of the software component of a safety-critical system. As with 00-56, the criticality can be used to moderate the strength of argument and evidence required to assure the safe behaviour of the software.

There is no direct equivalent to the 00-56 requirement for the production of Safety Cases and Safety Case Reports in MilStd 882E. 882E does talk about the production of a “technical data package”. However, this seems to be a collation of the project safety evidence, e.g. test results and safety analyses, rather than an explicit presentation of safety *arguments* and evidence as required by a Safety Case Report in the UK.

In practice, differences in UK and US practice in the area of safety cases do cause project difficulties. In particular, the UK expectation of the content, scope and depth of a safety case report is unfamiliar to many US system suppliers. However, there are an increasing number of instances of where US suppliers are beginning to produce safety cases (for systems exported to the UK and for joint US-UK initiatives).

The US, with UK Ministry of Defence sponsorship, is looking to adapt MilStd 882 to remove the differences/risks currently perceived as existing between these standards. The aim being worked towards is to have an "industry version" of MilStd 882 which could meet both the "formal" requirements of MilStd 882 and UK Defence Standard 00-56 Issue 4.

2.2.1 Applying Product-Based Assurance Approaches to Adaptive Systems Justification

A product evidence-based approach to the assurance of the safety of adaptive systems demands the production of safety arguments that are hazard and risk focused. A safety case is required that identifies and addresses failures of the adaptive system that can be shown to contribute to system level hazards. A product evidence-based approach is not primarily concerned with how an adaptive system has been *developed*. Instead, it is concerned with *operational behaviour* of the system. Whilst this is ultimately a more direct and compelling approach to the justification of safety, it means that the arguments of safety will be forced to engage in the details of the mechanisms and stimuli that lead to (potentially unsafe) changes in the behaviour of an adaptive system.

Both the concept of software ‘criticality’ in 882E and the notion of moderating arguments and evidence according to the risk associated with system operation (from 00-56) have bearing on the safety cases required for adaptive systems. Where adaptive systems are placed in low-criticality (e.g. advisory) roles, the strength of argument required is low. It may even be possible to rely upon inductive statistical evidence concerning the performance of the adaptive system (e.g. as suggested by approaches such as [11]). However, if adaptive systems are to be placed in high-criticality applications (e.g. with direct autonomous control over safety-critical functions) then strong, deductive and *analytical* arguments and evidence will be required.

There have been previous examples where testing has been shown to provide inadequate evidence of the safety of a software system, and where analysis has been shown to be preferable. An example of this can be seen in scheduling for safety-critical control systems, with the move from cyclic executive schedulers to fixed-priority task structure schedulers. Fixed priority schedulers have many desirable properties when compared to cyclic executives, but their behaviour is not deterministic and therefore they are much less amenable to testing. Analysis is therefore needed in order to provide adequate guarantees of their real-time performance [12].

3. Safety Case Patterns for Adaptive Systems

The previous report [1] outlined out possible approaches to arguing the safety of adaptive systems. The earlier analysis in the project indicated that a product-focused argument would be needed. The report set out some outline arguments necessary to provide this product-focused argument. However, no "joined up" argument was produced. This section of the report expands upon these original arguments and documents them as fourteen argument patterns described using the Goal Structuring Notation (GSN). The aim in presenting this collection of patterns is to indicate how the overall argument of acceptable safety can be decomposed to the point where the necessary claims regarding the technical characteristics of an adaptive system are clearly identified.

Before presenting the patterns in sections 3.2 to 3.11, the following section presents a brief overview of the use of GSN to present safety arguments, and safety argument patterns.

3.1 Using GSN to Present Safety Arguments

The Goal Structuring Notation (GSN) [13] – a graphical argumentation notation – explicitly represents the individual elements of any safety argument (requirements, claims, evidence and context) and (perhaps more significantly) the relationships that exist between these elements (i.e. how individual requirements are supported by specific claims, how claims are supported by evidence and the assumed context that is defined for the argument). The principal symbols of the notation are shown in Figure 4 (with example instances of each concept).

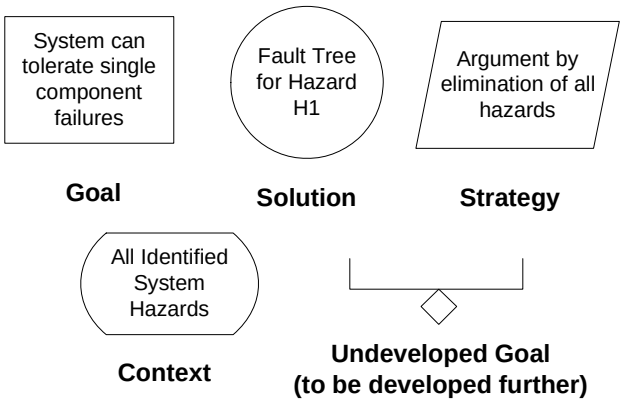

Figure 2 – Principal Elements of the Goal Structuring Notation

When the elements of the GSN are linked together in a network they are described as a 'goal structure'. The principal purpose of any goal structure is to show how goals (claims about the system) are successively broken down into sub-goals until a point is reached where claims can be supported by direct reference to available evidence (solutions). As part of this decomposition, using the GSN it is also possible to make clear the argument strategies adopted (e.g. adopting a quantitative or qualitative approach), the rationale for the approach and the context in which goals are stated (e.g. the system scope or the assumed operational role).

Figure 3 shows an example goal structure for a conventional control system. In this structure, as in most, there exist 'top level' goals – statements that the goal structure is designed to support. In this case, "C/S (Control System) Logic is fault free", is the (singular) top level goal. Beneath the top level goal or goals, the structure is broken down into sub-goals, either directly or, as in this case, indirectly through a strategy. The two argument strategies put forward as a means of addressing the top level goal in Figure 5 are "Argument by satisfaction of all C/S (Control System) safety requirements", and, "Argument by omission of all identified software hazards". These strategies are then substantiated by five sub-goals. At some stage in a goal structure, a goal statement is put forward that need not be broken down and can be clearly supported by reference to some evidence. In this case, the goal "Unintended Closing of press after PoNR (Point of No Return) can only occur as a result of component failure", is supported by direct reference to the solutions, "Fault tree cutsets …" and "Hazard Directed Testing Results".

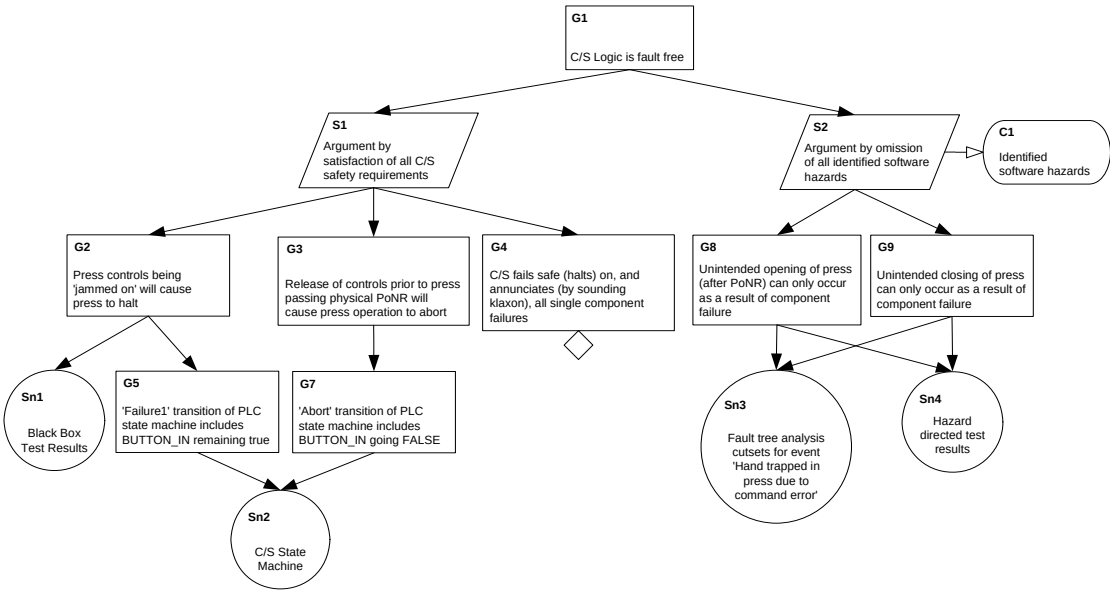

Figure 3 – An Example Goal Structure (for a conventional system)

A number of extensions have been made to GSN to support express of *generalised / abstracted* safety argument *patterns*. Figure 4 shows a simple goal structure pattern that uses these extensions. In this structure, the top-level goal of system safety (*G1*) is re-expressed as a number of goals of functional safety (*G2*) as part of the strategy identified by S1. In order to support this strategy, it is necessary to have identified all system functions affecting overall safety (*C1*) e.g. through a Functional Hazard Analysis. In addition, it is also necessary to put forward (and develop) the claim that either all the identified functions are independent, and therefore have no interactions that could give rise to hazards (*G4*) or that any interactions that have been identified are non-hazardous (*G3*).

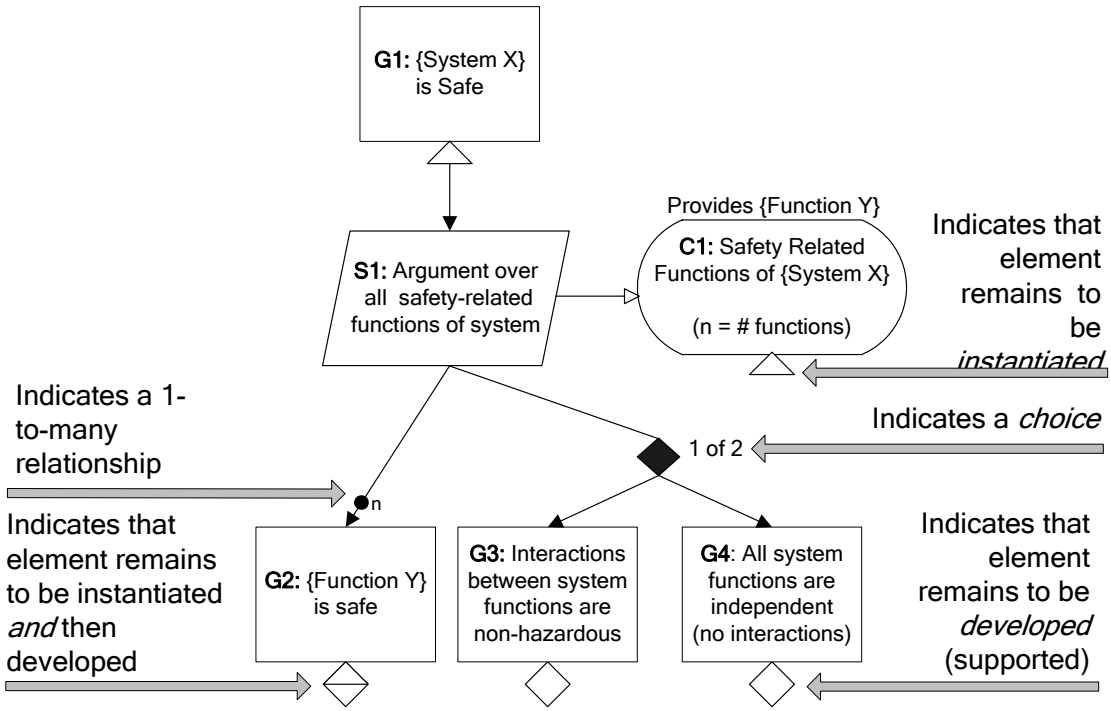

Figure 4 – GSN Extensions for Pattern Description

The following sections (3.2 to 3.11) use GSN, and the extensions for presenting generalised / abstracted arguments, to present fourteen safety case argument patterns that can be used to establish the principal arguments of safety required for an adaptive system.

3.2 Improved or Maintained Safety

The first argument pattern deals with the distinction (as discussed in Section 1 of this report) as to whether we have introduced an adaptive capability to improve safety or to improve some other system attribute (e.g. performance). This distinction can govern whether safety is viewed as an *objective* or a *constraint*, and impacts upon the top-level claims of any safety argument. Figure 5 outlines the structure of the top-level argument.

Regardless of the motivation for introducing adaptation, the goal *Top* must be supported. As with any such top-level claim, context such as the adaptive system definition (*SysDefn*) and the operating region over which the system is considered to be safe (*SafeOpReg*) must be provided. *SafeOpReg* defines the entire space of normal and abnormal operating conditions under which the adaptive system will continue to operate safety.

Underneath *Top* we see the choice in supporting claims that is associated with the motivation for introducing safety. If we have introduced adaptation to improve safety *ImpSaf* represents the claim that should be made. If adaptation has been introduced for reasons other than safety *AdaptNotUnacc* should be stated. *AdaptNotUnacc* states that safety is maintained even in the presence of introducing an adaptation capability. There are two possible means of supporting this claim. One approach is to argue by comparison. To do this, a definition of the 'conventional' (non-adaptive) system being used as the basis comparison needs to be referenced by *ConvSys.* (As shown in the pattern, this is also required for the argument of improvement of safety *ImpSaf*.) An alternative to comparison is to argue by reference to addressing defined risk acceptance criteria that exist (*AdaptRiskAcc*).

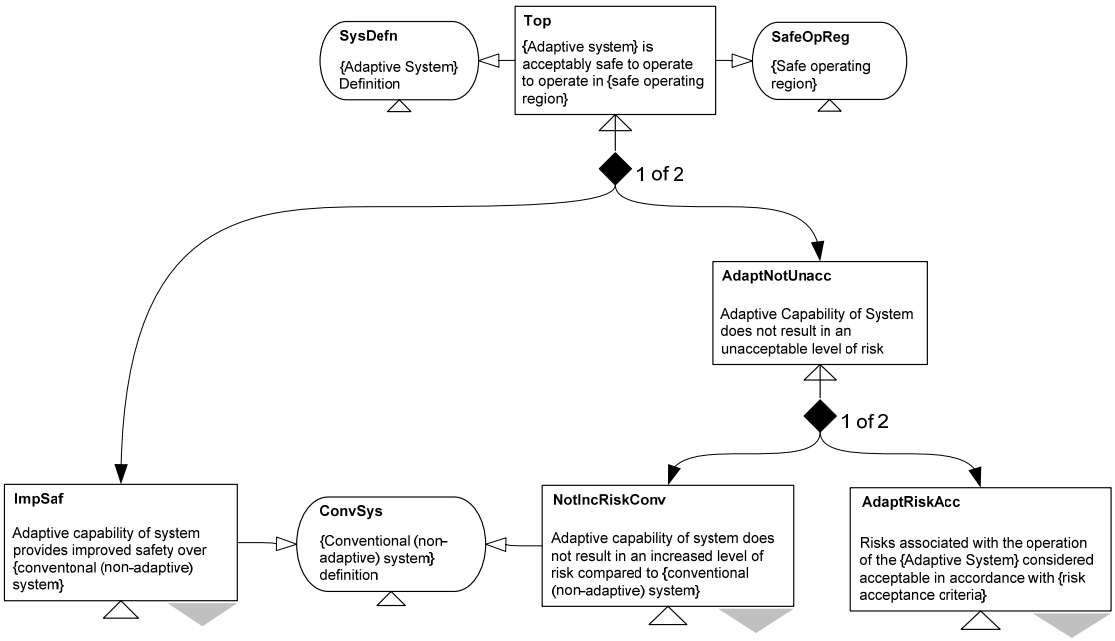

Figure 5 – Improved or Maintained Safety Argument

3.3 Improved Safety

Continuing from the previous pattern, Figure 6 shows the argument pattern for supporting the claim that the adaptive capability of the system has improved safety (*ImpSaf*).

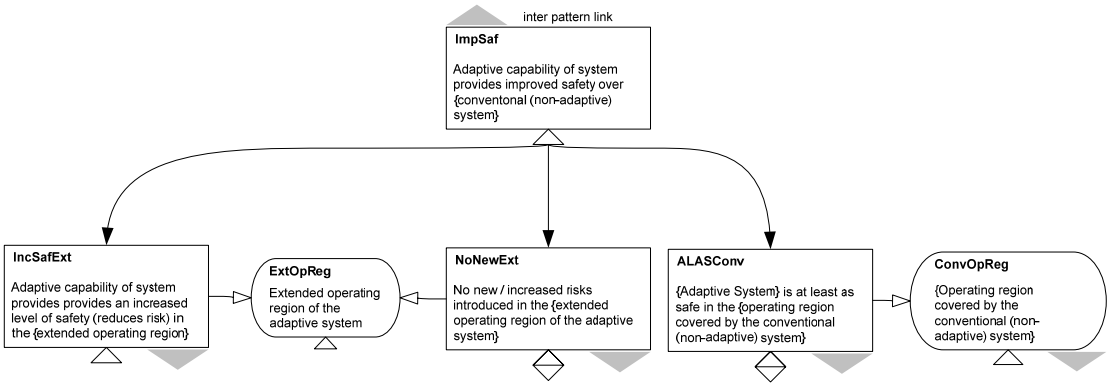

Figure 6 – Improved Safety Argument

The argument of improved safety implies that the operating region of the system has been extended – i.e. that there were operating conditions that would have previously led to an unsafe state, and that are now addressed by the introduced adaptive capability. It is important to define clearly *how* the operating region of the system has been extended, when compared to the operating region of the conventional (non-adaptive) system. This is achieved through instantiating *ExtOpReg* and *ConvOpReg*.

As shown, there are three elements to the argument of improved safety. Firstly, that the adaptive system is helping to reduce risk in the extended operating region (*IncSafExt*). Secondly, that no new or increased risks are present in the extended operating region

(*NoNewExt*). Finally, the increased safety in the extended operating region cannot be at the expense of reduced safety in the operating region previously addressed by the conventional (non-adaptive) system. This argument is captured in claim *ALASConv*.

3.4 Maintained Safety

The argument of maintained safety is a subset of the argument of improved safety discussed in the previous section. This is the essential argument required when adaptation has been introduced for reasons other than safety (e.g. to improve performance or availability or to reduce costs). Figure 7 outlines the required components of this argument.

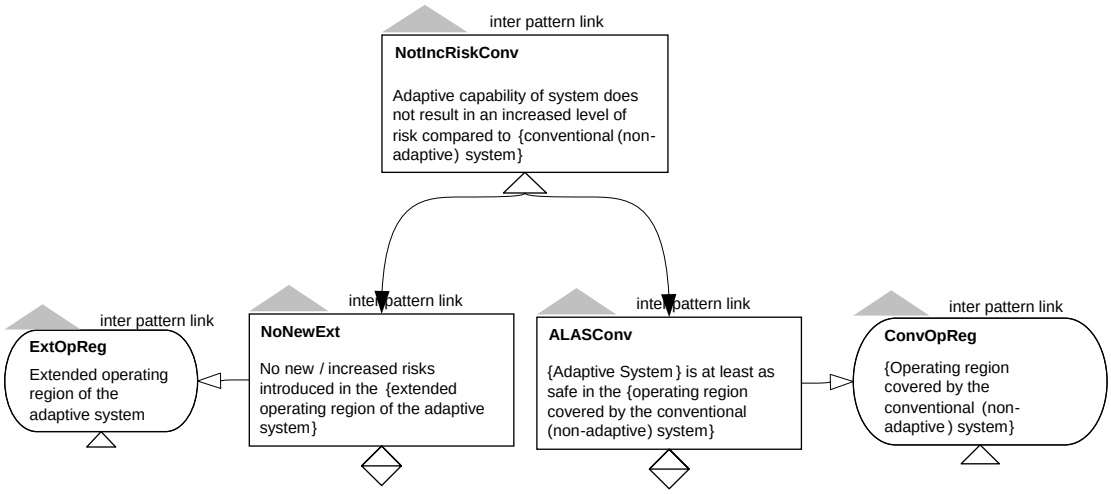

Figure 7 – Maintained Safety Argument

The adaptation capability of the system has extended the operating region of the system (*ExtOpReg*). The first concern is that there are no increased risks in this extended operating region (*NoNewExt*). Secondly, we are concerned that the adaptive system is at least as safe as the 'conventional' system *(ALASConv*) over the previously covered operating region (*ConvOpReg*). In the next section we use an existing argument pattern to illustrate the required elements of an argument capable of supporting *ALASConv*.

3.5 At Least As Safe

Figure 8 shows an existing pattern that illustrates the challenges that must be addressed when attempting to argue that one system is at least as safe as another. The top two layers of this pattern overlap with the arguments already presented in the previous sections (specifically *NotIncRiskCov, AdaptNotUnacc,* and *Top*). (Note – the diamond symbol represents a contextual reference to system model information.) The argument required to support *ALASConv* is shown in the strategy *AtLeastAsSafeArg* and its supporting elements.

In the absence of acceptability targets / criteria for the 'new' adaptive system, an obvious minimal requirement is that overall safety is not worsened by the introduction of the new system. (In general, it is usually desired that the overall level of safety is either the same or improving over time – deterioration is not usually accepted).

Importantly, to carry out this argument strategy it is essential that the safety record of the system being replaced is known, otherwise comparison will be impossible. This context reference *ExistSysSafetyRecord* must be instantiated to point to the where the safety record of the existing system can be found. The credibility of this argument approach hangs upon the integrity and completeness of this evidence.

A reference is made by the pattern to a description of the 'conventional' system being replaced *ExistSysDesc*. The 'at least as safe' approach creates a weak argument if it is not possible to justify that the existing system was safe *ExistSysAccSafe*. Further, the 'at least as safe' approach is only valid if the two systems (existing and new) are comparable (*SimilarSys)*. It is important to establish the criteria (*SuffSimilarDefn)* by which it shall be judged if the new and existing systems are sufficiently similar.

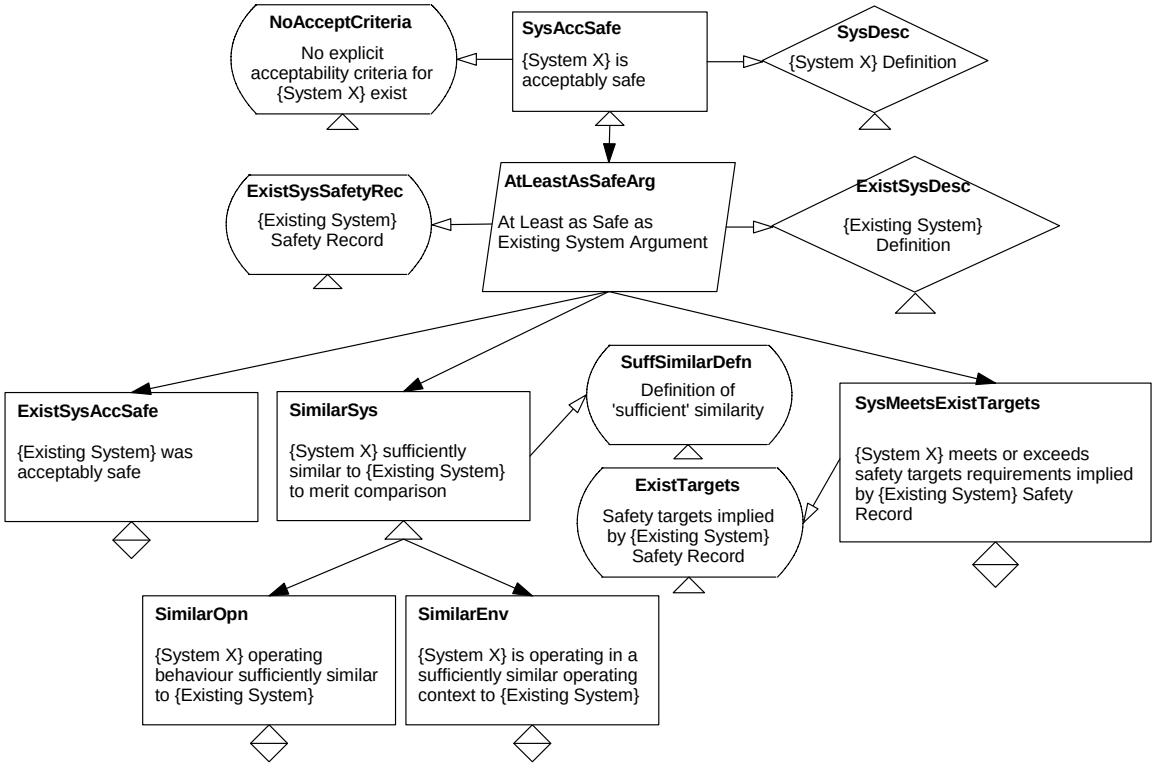

Figure 8 – At Least As Safe Argument

The main claim of the 'at least as safe' approach is *SysMeetsExistTargets*. It is necessary to instantiate and support the claim that, based upon criteria implied by the existing system's safety record, the new (adaptive) system either meets or exceeds the safety of the existing system. For this to work it must be possible to instantiate *ExistTargets* with reference to requirements derived from the operational safety record of the existing system. These requirements could be both *quantitative* (e.g. an acceptable rate of occurrence for a hazard) or *qualitative* (e.g. the absence of a particular failure mode, or the presence of a hazard mitigation behaviour). It is worth recognising that complete back-to-back verification of the new system against the existing system is not necessarily required. Instead, this argument requires (through instantiating *ExistTargets*) that we are able to 'distil' the essential characteristics of the operational behaviour of the existing system to use as the 'benchmark' for evaluation of the new system.

3.6 Risk Acceptance

As discussed in section 3.4, an argument of acceptable safety does not necessarily require comparison with existing systems. Figure 9 shows the alternative approach of arguing acceptable safety by reference to (externally) stated risk acceptance requirements.

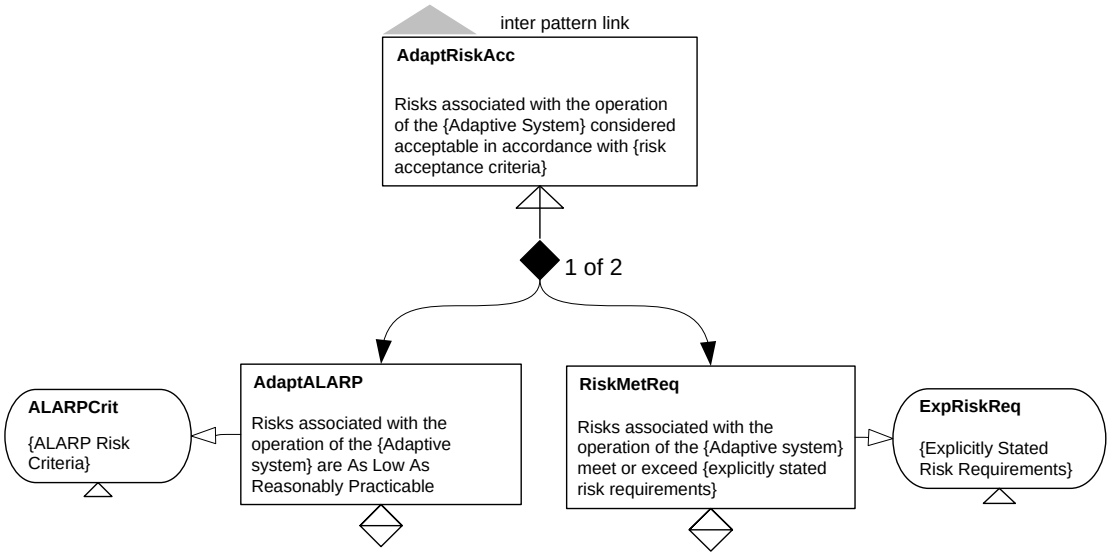

Figure 9 – Risk Acceptance Argument

Two alternatives claims are shown in Figure 9. The first (*RiskMetReq*) is that risks associated with the adaptive system meet or exceed some explicitly stated risk requirements (e.g. an overall level of tolerable risk, or an acceptable Hazard Risk Index). The second approach (*AdaptALARP*) is potentially of more value for adaptive systems. This claims that the risks associated with the operation of the system as 'As Low As Reasonably Practicable'. As discussed in the next section, the ALARP principle allows acknowledgement of the benefit associated with the risks being posed, and the costs of risk reduction, to be included in the discussion of risk acceptance.

3.7 As Low as Reasonably Practicable

If risks are introduced or increased by the introduction of an adaptive capability, it may be necessary to argue that these risks are necessarily present, and acceptably controlled. The argument pattern shown in Figure 10 provides a framework for arguing that identified risks in a system have been sufficiently addressed in accordance with the ALARP principle. This pattern could be used to support the *AdaptALARP* claim from the pattern described in the previous section.

The ALARP principle divides risk into three categories. Firstly, there are *intolerable* risks that cannot be justified on any grounds. At the other extreme, there are risks that are sufficiently low to be considered *negligible*. Between these two extremes (in the ALARP region) there are potentially *tolerable* risks. The tolerability of these risks depends on whether they can be argued to be *necessarily present* and that they have been reduced to the point that further risk reduction would cost an amount *disproportionate* to the improvement gained.

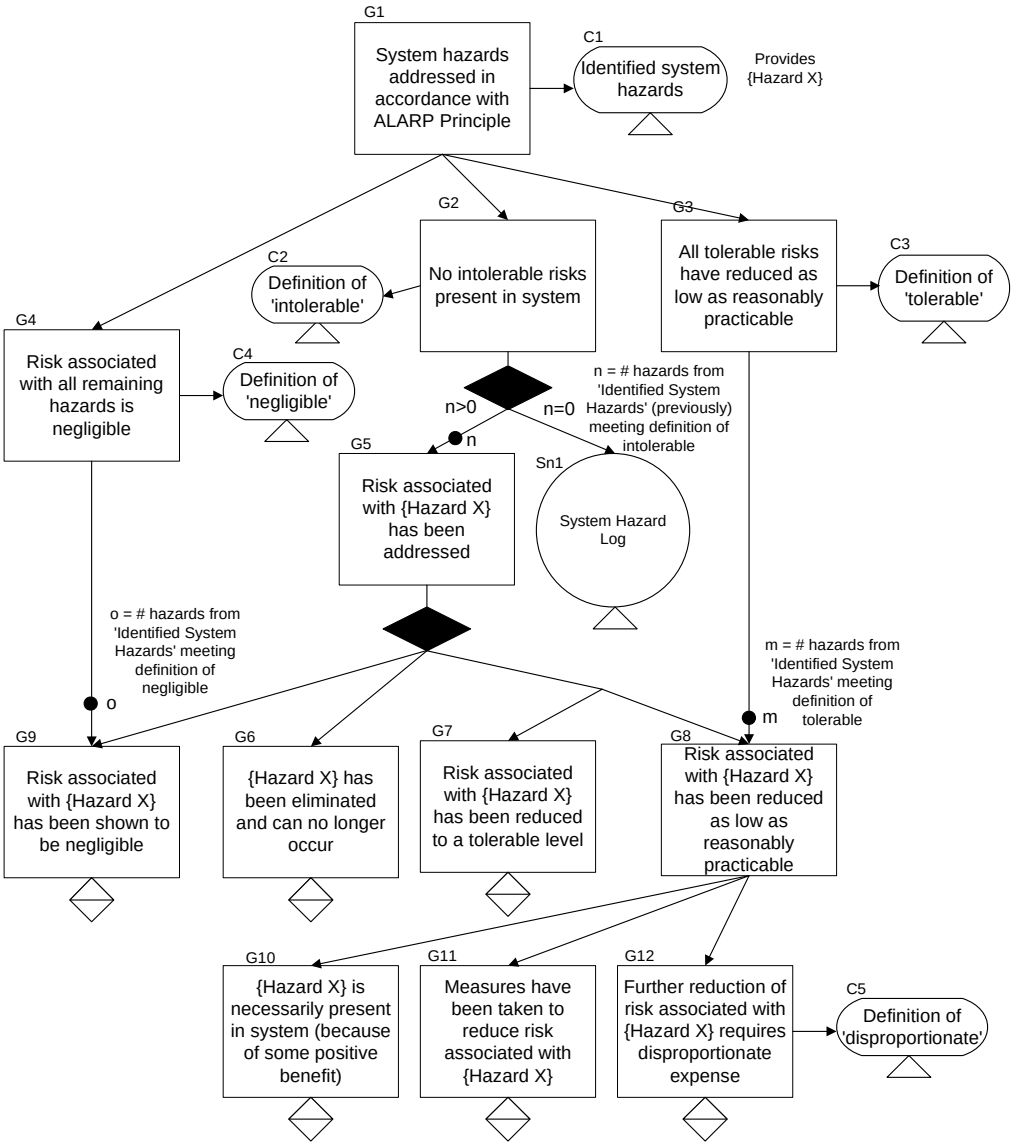

Figure 10 – ALARP (As Low as Reasonably Practicable) Argument

There are three strands to the safety argument: one tackling intolerable risks, one tackling tolerable risk and one discounting negligible risks. To satisfactorily support the top level goal (*G1*) it is important that these three strands (collectively) address all identified risks. The definitions of tolerable, intolerable and negligible (*C3, C2* and *C4* respectively) should therefore be so defined to cover and classify the range of possible levels of risks.

The ALARP principle relies on some understanding (*C5)* of when it is no longer cost-effective to spend further money on risk reduction. The definitions of negligible (*C4*) and disproportionate (*C5*) cannot be considered entirely independently. It would not make sense, for example, to force risk reduction to a level below that identified elsewhere as negligible.

As the goal structure shows, if the means of addressing a previously identified intolerable risk is to reduce it to a tolerable level, then the remaining risk must be tackled as for all tolerable risks. If the level of risk has been reduced to a negligible level, then the hazard must be tackled as a negligible risk.

It is important that the source of Identified System Hazards (*C1*) identifies the level of risk posed by a hazard in a way that permits sub-division into the classes of risk defined by *C2, C3* and *C4*.

Ultimately, the ALARP pattern requires supporting arguments for the claims of elimination (absence) of intolerable risks (*G6*), acceptably low occurrence of negligible risks (*G9*), and the measures taken to reduce tolerable risks (*G11*).

3.8 Mapping High Level Hazard Control / Risk Reduction Claims to the Safety Features of the Adaptive System

Working top-down through the arguments presented so far, a number of leaf goals remain that are expressed in (general) terms of overall risk levels. For example, consider the following leaf goals:

- (From the "Improved Safety" pattern) *IncSafExt* - Adaptive capability of system provides an increased level of safety (reduces risk) in the {extended operating region}
- (From both the "Improved Safety" and "Maintained Safety" patterns) *NoNewExt* - No new / increased risks introduced in the {extended operating region of the adaptive system}
- (From the "At Least As Safe" pattern) *SysMeetsExistTargets* – {System X} meets or exceeds safety targets implied by {Existing System} Safety Record
- (From the "ALARP" pattern) *G7* - Risk associated with {Hazard X} has been reduced to a tolerable level.

As discussed in Section 2, *process-assurance* based arguments could possibly be used to support these claims. However, given the *specific* (hazard and risk oriented) nature of these claims, a *general* appeal to the integrity of the development and V&V processes would be far from compelling. (Why should someone believe, for example, that no new / increased risks have been introduced in the extended operating region of the adaptive systems simply because the adaptive system has been developed to DO-178B Level A?) Instead, a *product-based* approach suggests that these claims should be decomposed to claims and evidence regarding the technical characteristics of the adaptive system in question.

For the *positive* argument of increased safety through adaptation (*IncSafExt*) we must decompose this claim to the specific contributions of the adaptive software system to hazard mitigation. This decomposition is shown in Figure 11.

To argue improved safety it is necessary to argue that there were risks previously present with the conventional (non-adaptive) system that are now reduced through the introduced adaptation capability (*AdaptRedRisks*). This risk reduction claim can be addressed though arguing the acceptably implementation of hazard mitigation requirements (*ArgOverPosReq*), with the requirements clearly referenced by *PosRiskRedReq*. As with any such mapping of risks to requirements, it is necessary to argue that the requirements are valid (*PosReqValid*) and traceable (*ReqTraceable*). Having established the requirement, it should be acknowledged that there can be a hardware and software contribution to acceptably addressing the requirements (e.g. a requirement on the reliability of the hardware platform as well as on the behaviour of the software). This is shown in the decomposition of strategy

ArgSWHWOtherPos. (The 'other' contribution mentioned in *OtherContribPosAccept* acknowledges that other systems, including humans, can also contribute to the achievement of a hazard mitigation requirement.)

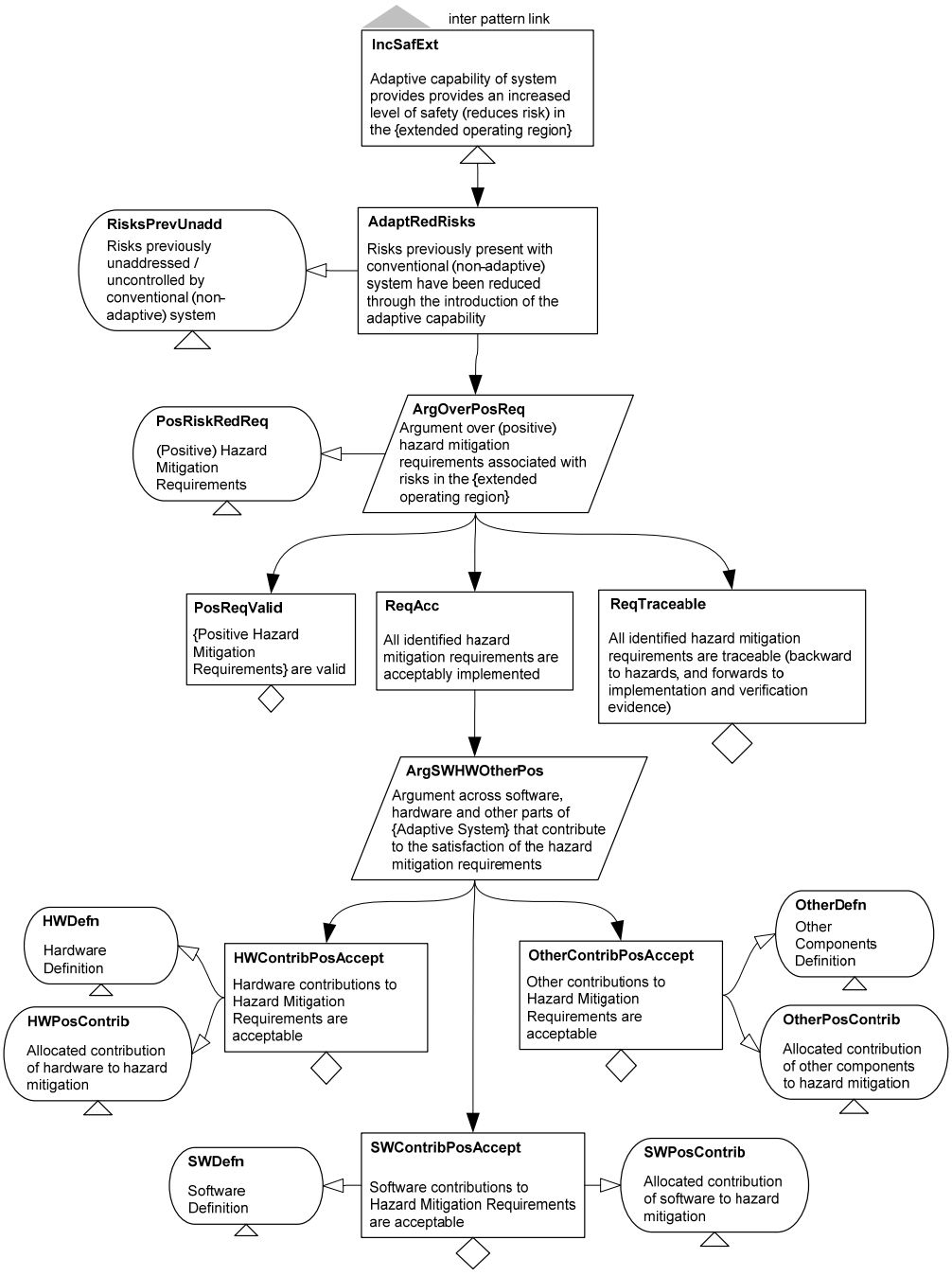

Figure 11 - Top Level System-to-Software Hazard Mitigation Argument

Importantly, for the purpose of producing a product-oriented software safety argument, we are left with the goal *SWContribPosAccept*. This goal states that adaptive software system acceptably 'plays its part' in the implementation of hazard mitigation.

3.8.1 Top Level System-to-Software Hazard Contribution

For the *negative* arguments that risks associated with the operation of the adaptive system are acceptably low (e.g. *NoNewExt*) we must decompose these claims to point where specific contributions of the adaptive software system to system level hazards can be argued to be absent. Weaver's [14] software safety argument presents such a decomposition, and is shown in Figure 12.

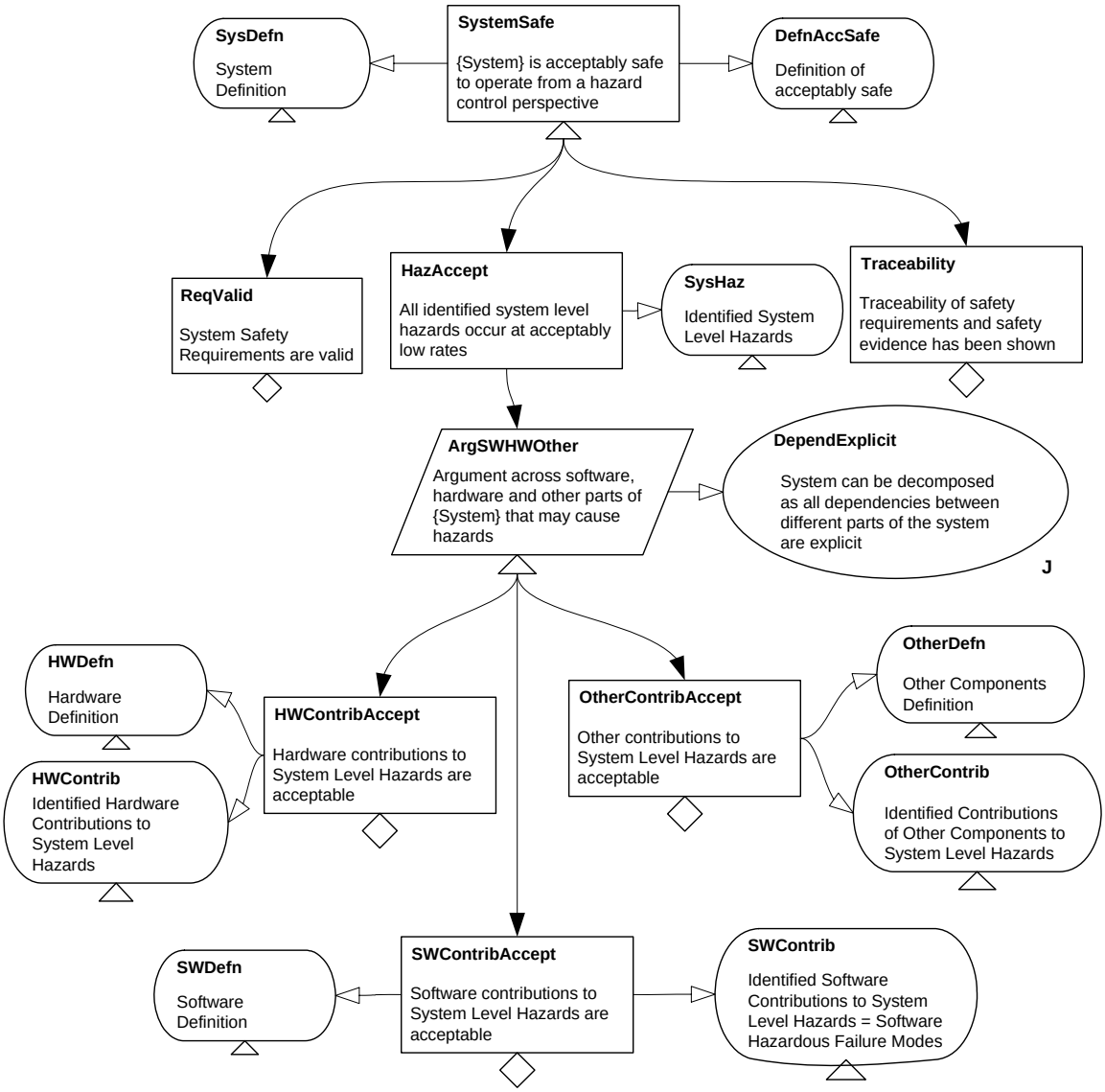

Figure 12 - Top Level System-to-Software Hazard Contribution Argument (from [14])

This pattern follows a similar structure to the positive decomposition shown in Figure 11 – the key difference being that we considering how hazards internal to the adaptive system can arise (and arguing the acceptable non-occurrence of these conditions) rather than addressing how hazards external to the adaptive system can be controlled.

Again, importantly, for the purpose of producing a product-oriented software safety argument, we are left with the goal *SWContribAccept*. This goal states that the potential contributions of the adaptive software system to system-level hazards are acceptably controlled.

To decompose the argument further it is necessary to consider the detail of how software level contributions can be identified and caused by the software system. Such a decomposition is shown by Weaver's existing pattern Figure 13.

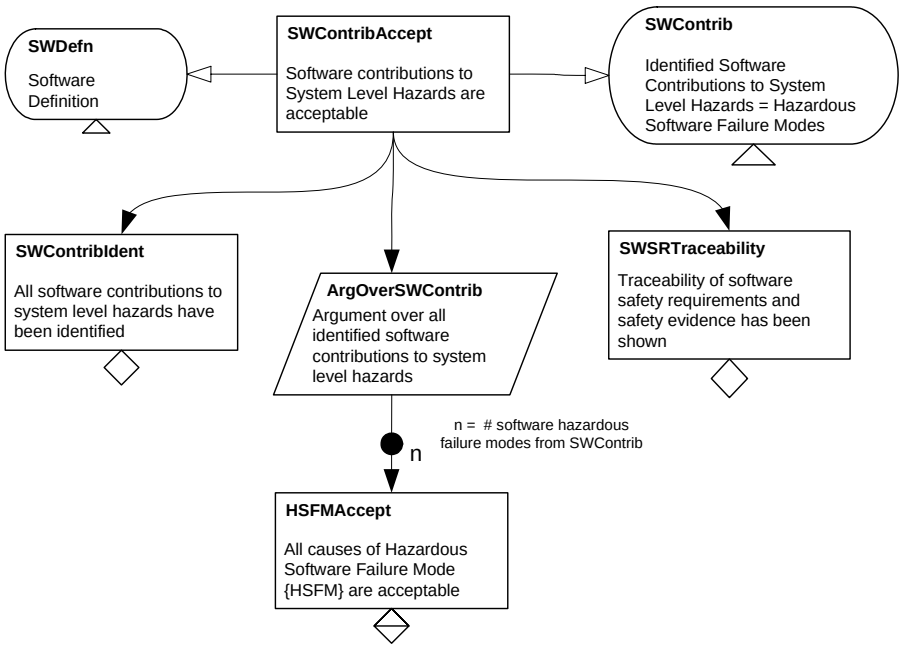

Figure 13 - Software Hazard Contributions Argument (from [14])

This pattern was originally developed for conventional software systems. Whilst it remains valid for adaptive systems, it is important to recognise that for adaptive system, satisfactorily arguing that all software contributions have been identified (*SWContribIdent*) and developing the argument over all software contributions (*ArgOverSWContrib*) could be particularly challenging. As already discussed, it is possible that with an adaptive system, the functional behaviour may be only partially defined at the outset of development. Behaviour could be learnt through a process of training. Any learnt behaviour must be examined to identify whether, and how, it can contribute to system level hazards.

The identification of unsafe conditions of an adapting software system was a core concern of the Safety Lifecycle for Artificial Neural Networks (SLANN) developed by Kurd [15]. This lifecycle encapsulates several development and safety tasks involved in generating adaptive control systems. At the initial phases of development the domain experts and safety engineers may find it difficult to accurately define the desired function, and the degree to which the adaptive control system can be allowed to adapt. The SLANN approach addresses cases when there is potentially incomplete and incorrect functional specification at the initial phases of development. Following an evolutionary style approach, the SLANN exploits prior knowledge (in the form of rules that attempt to describe the desired function) gathered from domain experts and 'insertion' algorithms [15] to generate a Safety-Critical Artificial Neural Network (SCANN). Following an iterative approach, the SLANN further develops the adaptive control system using learning processes. During the learning process existing rules may be tuned and new rules self generated. An advantage of this approach is that the development of the adaptive control system will take place by directly interacting with the environment or problem in which it intends to operate (such as a simulation). This direct interaction can help address potentially unrealistic assumptions the developers may make. Safety assessment processes use rules that have been extracted from the SCANN and analyses them to determine whether the approximated function exhibits any identified failure modes.

As with conventional software systems, it is possible to classify the types of hazardous software failure mode that could arise from the Contributing Software Functionality (CSF). Pumfrey's classification [16] uses a service-based classification of failures. The classification is refinement of the classifications by Ezhilchelvan and Shrivastava, and Bondavalli and Simoncini. This service based approach identifies five different types of failures:

- **Omission:** The service is never delivered
- **Commission:** A service is delivered when not required
- **Early:** The service occurs earlier than intended
- **Late:** The service occurs later than intended
- **Value:** The output value has the wrong value

By defining the type of failure mode of concern, it is possible to focus the argument on the particular causes and associated evidence for that type. This is shown in Figure 14.

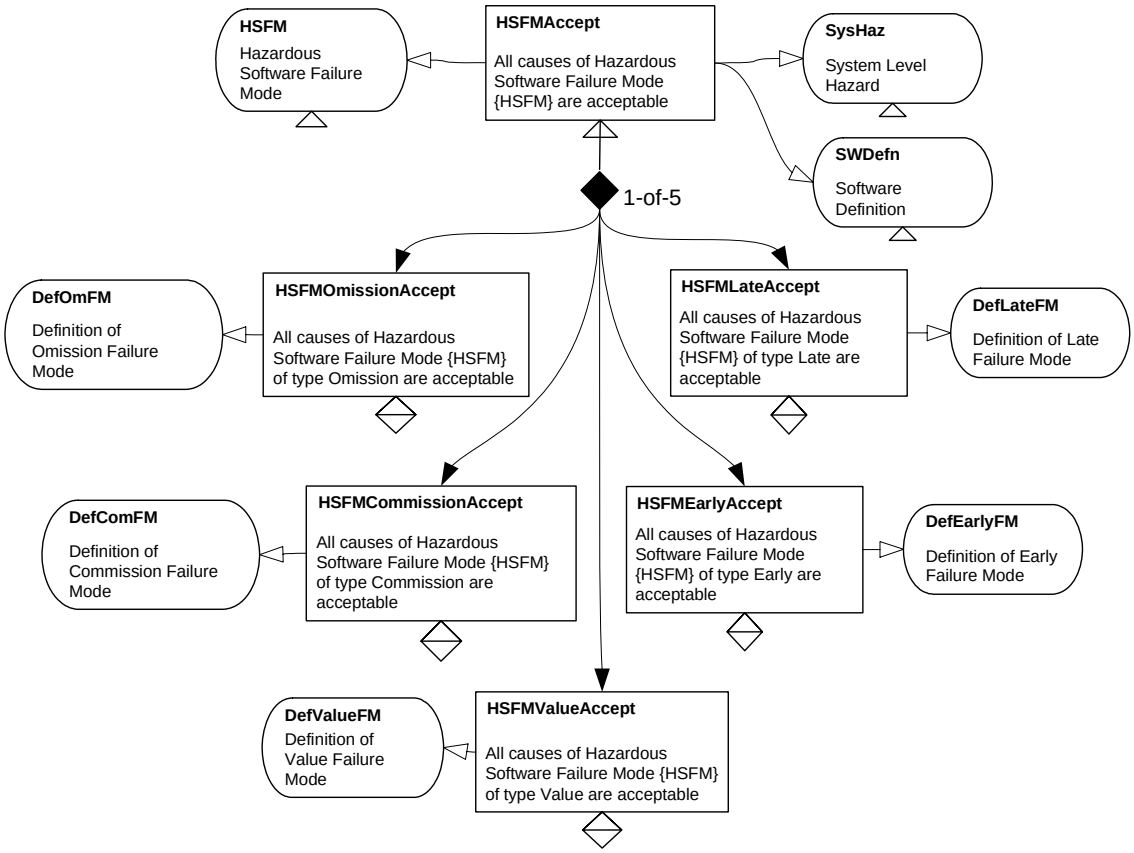

Figure 14 – Hazardous Software Failure Mode (HSFM) Classification Argument (from [14])

The classification of software failure modes can be useful for adaptive systems. Kurd in [15] shows how HAZOPS (Hazard and Operability Study) guidewords can be interpreted when applied to an Artificial Neural Network used for a function approximation problem. For example, the guideword 'NONE' or 'No' is interpreted as follows:

> *Data value, 'NO': The 'NO' guide word was interpreted as ''no output signal'' or output omission. This interpretation for the SCANN [Safety Critical Artificial Neural Network] specifically became ''no output data given an input vector''.*

Kurd goes on to show in [15] how the specific features of the adaptation mechanism and process of the SCANN can be used to argue the absence of this failure mode.

The final part of mapping the claims that a software failure mode is 'acceptably' addressed is shown in Figure 15 (again, taken from Weaver's existing pattern catalogue [14]).

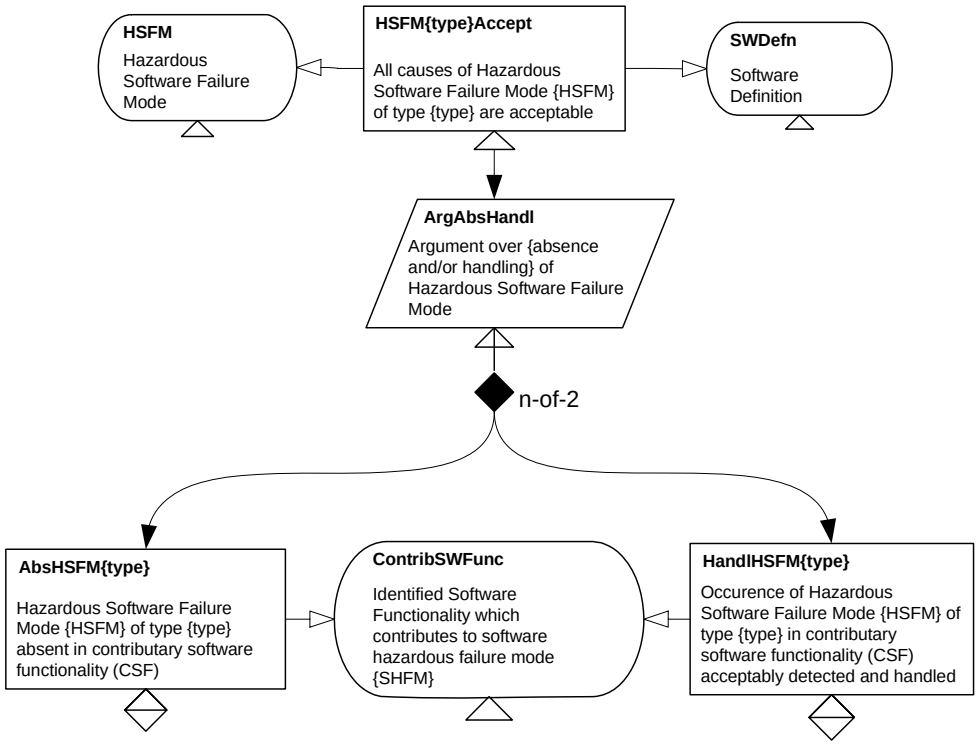

Figure 15 – Hazardous Software Failure Mode Acceptability Argument (from [14])

In this argument the failure mode acceptability claim (*HSFM{type}Accept*) is addressed by either arguing that the failure mode is *absent* from the adaptive software functionality (*AbsHSFM{type}*), or by arguing that the failure mode is acceptably detected and handled by some capability external to the adaptive software functionality (*HandlHSFM{type}*). (By the n-of-2, rather than 1-of-2 choice) the pattern also admits a third option – the use of both supporting arguments (i.e. that the failure mode is absent, but even if it were somehow present it would be detected and handled).

In the domain of adaptive systems, the use of a detection and handling argument such as *HandlHSFM{type}* relates to the potential use of external monitoring of, and back-up to, the adaptive system (where the back-up could be a conventional system). One such example of this is described in [17]. In this example the inputs into the adaptive control system are also fed into the 'monitor'. In addition, the output of the adaptive control system is also fed into the 'monitor'. The monitor then uses rules and algorithms to examine each input and output whilst the adaptive control system is in operation (i.e. aircraft surface control). One goal of the monitor is to identify output failures associated with the adaptive system (i.e. if the output is too high or too low for the given inputs) before the control signal is sent to the equipment under control. If no failure is detected then the output is not suppressed. If the monitor detects a failure mode in the output then the monitor can 'switch' to alternative conventional control software. This control software uses conventional control algorithms and it is assumed that one exists for the operating context in which the adaptive system will be used. The conventional control system is then used to 'take-over' from the adaptive

system until the 'monitor' detects that the output of the adaptive control system no longer results in an identified failure mode. In this example, the assurance level or Safety Integrity Level [2] will be applicable to the monitor, the switch and the conventional control algorithm. However, the adaptive control system can be of arbitrary assurance level [15]. This is because the main thrust of the safety argument does not include claims about the adaptive control system output since the 'final' output sent to the equipment under control will not result in identified failure modes.

One advantage of the combination of monitoring and conventional control software is that the approach can be applied for rapid prototypes. This can allow analysis and trials of adaptive control systems (of differing paradigms) and they can be efficiently assessed for performance. By involving monitoring approaches it can be less technically challenging to derive proofs and mathematical arguments concerning the behaviour of the adaptive control system and its capability. Another advantage is that existing standards and guidelines can be used to develop such systems and is therefore more realistic to generate safety argument that claims compliance with internal software and safety standards. As a result, there is greater possibility of safety case acceptance of AI based adaptive control systems.

It may not always be possible to provide an external monitor and back-up to the adaptive system. In such cases, the responsibility rests with the failure mode absence argument put forward by *AbsHSFM{type}.* The following section explores the possible arguments that can be used to support this claim.

3.9 Hazardous Software Failure Mode Absence

Figure 16 shows the pattern of argument that can be used to support the claim that a Hazardous Software Failure Mode is absent in the contributory software functionality of an adaptive system. Firstly, we are concerned about being able to argue that an initial state of the system cannot lead to the failure (*InitialHSFM*). Supporting this claim relies upon an ability to *know, interpret* and *analyse* the 'initial' state of the adaptive system. (In the case of Kurd's Safety Critical Artificial Neural Network, this involved being able to perform rule extraction to capture the rules implicit in the operating behaviour of the network.) It should be recognised that this initial *safe* state may not be the initial *development* state of the adaptive system. As described earlier, it can be desirable to develop adaptive systems in domains where the initial behaviour of the system is only partially defined, and the adaptation capability of the system is used to modify and add to this behaviour. Such an initial *development* state may already be hazardous (e.g. the hazard of omission arising from partially defined behaviour.)

Having argued the safety of the initial state of the CSF, it is then necessary to argue that adaptation of that CSF cannot lead to the introduction of the failure mode (*AdaptHSFM)*. To argue this claim requires detailed knowledge of the adaptation mechanism for the CSF (*AdaptMech*).

A distinction must be made when arguing the safety of any adaptation of the CSF as to whether the adaptation is allowed in operation (on-line learning) or whilst the system is out of operation (off-line learning). This is represented in the pattern as the choice between *OnLAdaptHSFM* and *OffLAdaptHSFM*. The key distinction between online and offline learning is that for off-line learning, we may be able argue that the behaviour that was derived by learning *is* not unsafe, whilst with on-line learning we must argue that the process of adaptation cannot *introduce* behaviour that is unsafe.

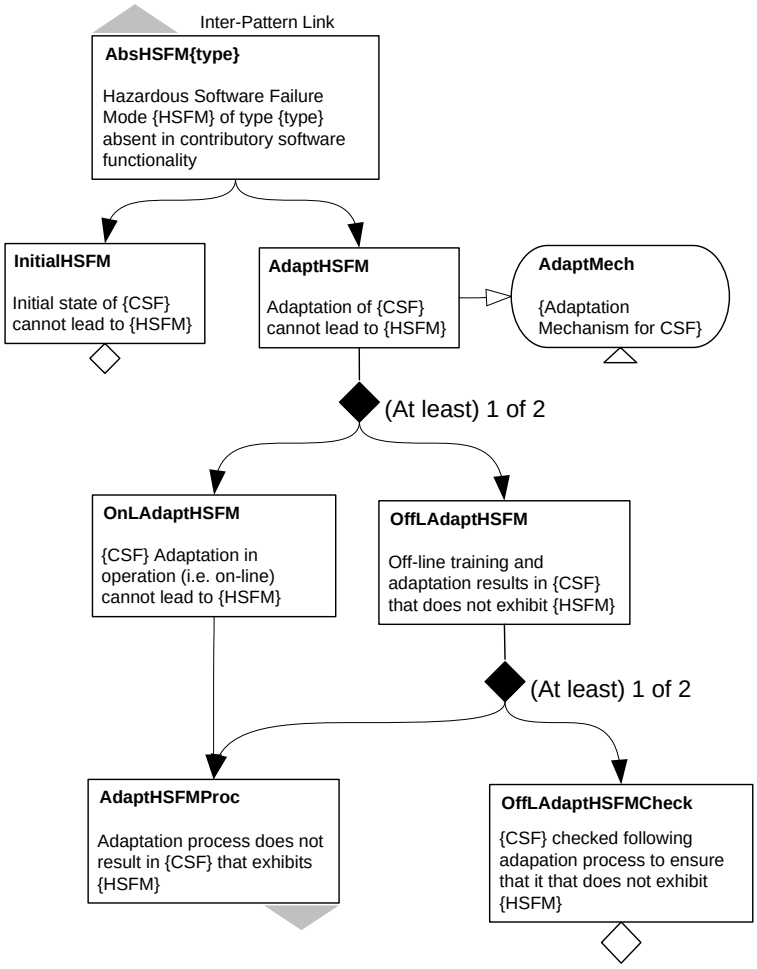

Figure 16 – Hazardous Software Failure Mode Absence Argument

For off-line learning the supporting goal *OffLAdaptHSFMCheck* suggests the possibility of checking the state of the CSF following any learning episode (just as was suggested for the initial state). The inability to check the adapted state of the CSF is a potential obstacle to supporting this goal. For example, Cukic, in [18], observes that the functional properties of an adaptive system cannot be inferred by a static analysis of the software; its functional properties at any time can only be derived from knowledge of both the system software and its dynamic state.

For an adaptive system employing *on-line* adaptation unsafe adaptations cannot be allowed to take place (at least not without undermining *AbsHSFM{type}*). The following section explores how it may be possible to argue the safety of the adaptation process (in support of *AdaptHSFMProc*).

3.10 Safe Adaptation

There are two possible supporting arguments to address the claim that the adaptation process cannot lead to the introduction of a hazardous failure mode (*AdaptHSFMProc)*. The first possible argument is that 'hazardous' adaptation stimuli (i.e. stimuli that could result in the introduction of the hazardous failure mode) are not presented to the adaptation mechanism. The principal difficulty in this argument lies in being able to determine the 'hazardous' set of stimuli (*UnsafeStimuli*). This problem is exacerbated where the *time order*

of the presentation of stimuli is key to determining the nature of the adaptation. In such cases, it is the *sequence* of stimuli that could give rise to the introduction of the failure mode that must be determined. To identify 'hazardous' stimuli requires significant understanding of the problem domain and of the behaviour of the adaptation mechanism. Although it is easiest to imagine the 'filtering' of adaptation stimuli for adaptation that takes place off-line (e.g. training set selection for a neural network), the goal *OffLAdaptHSFMStim* may also have a role for on-line adaptation. In operation, it may remain necessary to argue the integrity of any stimuli presented to the adaptation mechanism (e.g. if it is not able to 'safely' tolerate noisy or 'incorrect' stimuli). It may be possible to support *OffLAdaptHSFMStim* through masking of the stimuli allowed to be passed through to the adaptation mechanism. (This is analogous to the 'wrapper' concept [19] when using Component-Off-The-Shelf components known to malfunction when presented with certain inputs.)

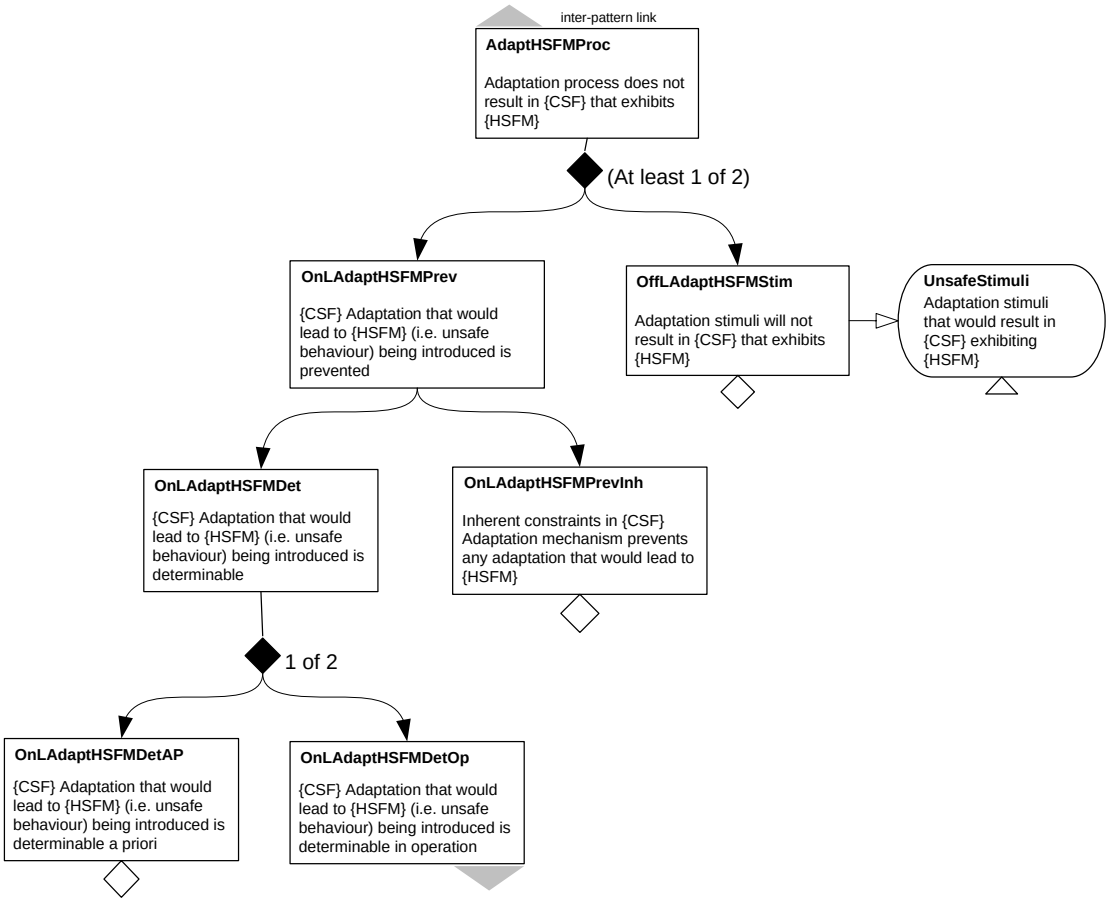

Figure 17 – Safe Adaptation Argument

The second approach to arguing the safety of the adaptation process is to argue that any possible 'unsafe' adaptation is prevented by inherent constraints within the adaptation mechanism (*OnLAdaptHSFM*). Kurd's SCANN is an example of an adaptation mechanism with in-built constraints [15]. The SCANN has a set of parameters which are used internally by functions to produce input-output mappings. The approach is to derive parameter constraints or safety constraints that control possible functions approximated by the SCANN. These safety constraints are devised through suitable safety and development processes to address identified failure modes associated with functional and non-functional properties. In addition, learning algorithms are also constrained to provide assurance that safety constraints are not violated.

The disadvantage of employing internal safety constraints is that they are more technically challenging to devise and enforce and are heavily dependant upon the adaptation mechanism. In addition, this approach requires that safety constraints are determinable (*OnLAdaptHSFMDet*) – i.e. that it is possible to determine when an unsafe adaptation is being prompted as part of the adaptation process. This argument could be supported either by claiming that static safety constraints are determinable prior to operation (*OnLAdaptHSFMDetAP*) or by claiming that safety constraints are determinable in operation. The latter of these two arguments is explored further in the next section.

3.11 Determining Unsafe Adaptations

The ability to determine unsafe adaptation in operation depends heavily upon the nature of the adaptation (learning) taking place. Consider two (non mutually exclusive) classes of adaptive system – model-building and behaviour-adapting. *Model-building* systems build a model over time using information from their environment. One example of this is SLAM (Simultaneous Localisation and Mapping), a process by which an autonomous vehicle can build a map of an unknown environment whilst simultaneously keeping track of its position on that map [20]. Whilst model-building systems acquire data over time, *behavioural adaptation* systems attempt to extract explicit patterns or rules automatically from the data acquired. An example of this is flight control system that modifies the way it uses its control actuators based on the response (in terms of flight behaviour) it detects in response to its previous use of the actuators. Such a system might, for example, might be able to continue normal flight in the event of an unanticipated actuator failure. In particular, it might manage to adapt to a failure mode that was not foreseen by the system designers.

For model-building systems, the behaviour of the system depends on the model it has constructed. This model is potentially built during operation. It is therefore extremely difficult to validate the model ahead of its use. As discussed in the previous section, it is necessary to argue that the system will not build a model (i.e adapt) in such a way as to introduce the potential for unsafe behaviour.

The argument shown in Figure 18 illustrates that the nature of the argument required to support the claim that unsafe adaptations can be determined during operation (*OnLAdaptHSFMDetOp*) is influenced by the nature of the adaptation taking place. For behavioural adaptation systems the claim *OnLBAdaptHSFMDetOp* could be supported by arguments of constraints enforced relative to the current system state, or even relative to other adaptations that have taken place. For model-building systems determining an unsafe model adaptation may be more difficult. Firstly, as shown in the pattern the model itself cannot be considered to be unsafe. To argue the safety of the model, it is necessary to consider the *uses* of the model (as shown by *ArgOverUC* and the context *CSFUC*). This results in claims that it is possible to determine the features of the model that could lead to HSFM being exhibited in the use of that model. In operation it is not possible to determine these ‘unsafe’ model features through comparison with ‘ground truth’. The best arguments that can be hoped for will depend on cross-check with other models available (and possibly constructed) in operation and/or checks performed using additional domain knowledge of the domain of the model (e.g. checking that a model constructed does not violate the laws of physics.)

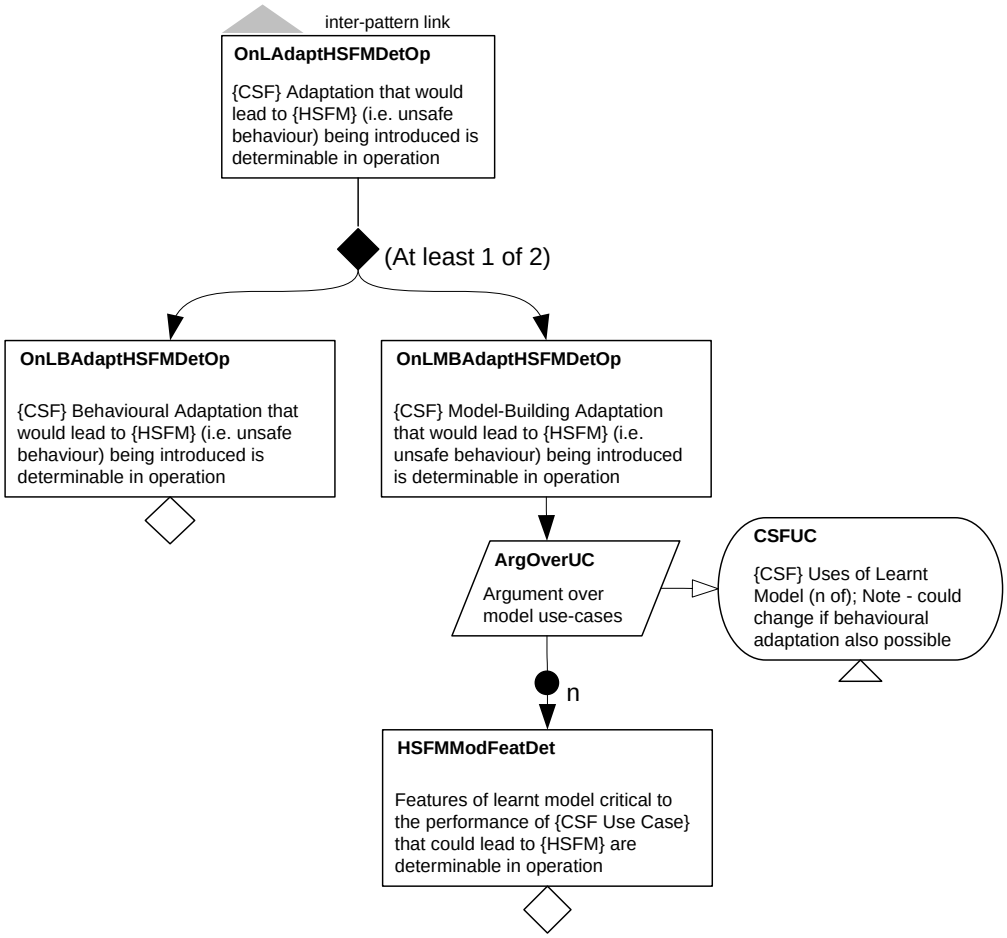

Figure 18 – Determining Unsafe Adaptations Argument

4. Summary

The novelty and perceived unpredictability of adaptive systems can make safety engineers and regulators look sceptically upon their potential use in safety-critical applications. This increases the need to establish compelling safety cases that assure their safe use in safety-critical applications. This report has provided an overview of existing (software) safety standards and has discussed the extent to which compliance with these standards will result in a compelling case for safety for adaptive systems. This discussion established the need to develop a product based approach to the justification of adaptive system safety. To illustrate the key safety arguments that would be required by such an approach, this report has presented a collection of the argument patterns – expressed using the Goal Structuring Notation (GSN). These patterns, if used together, can help establish the principal arguments of safety required for an adaptive system:

The following patterns have been presented in this report:

- Improved or Maintained Safety Argument
- Improved Safety Argument
- Maintained Safety Argument
- At Least As Safe Argument

- Risk Acceptance Argument
- ALARP (As Low as Reasonably Practicable) Argument
- Top Level System-to-Software Hazard Mitigation Argument
- Top Level System-to-Software Hazard Contribution Argument
- Software Hazard Contributions Argument
- Hazardous Software Failure Mode (HSFM) Classification Argument
- Hazardous Software Failure Mode Acceptability Argument
- Hazardous Software Failure Mode Absence Argument
- Safe Adaptation Argument
- Behavioural vs. Model-Building Adaptation Argument

5. References

[1] R. Alexander, M. Hall-May, T. Kelly and J. McDermid, Safety Cases for Advanced Control Software: Final Report, June 2007

[2] International Electrotechnical Commission (IEC), IEC61508 – Functional Safety of Electrical / Electronic / Programmable Electronic Safety-Related Systems, 1999

[3] Radio Technical Commission for Aeronautics (RTCA), RTCA DO-178B/EUROCAE ED-12B – Software Considerations in Airborne Systems and Equipment Certification, 1992

[4] UK Ministry of Defence, Defence Standard 00-56 – Safety Management Requirements for Defence Systems, Issue 4, 2007

[5] W. Wu and T. P. Kelly, Towards Evidence-Based Architectural Design for Safety-Critical Software Applications, in Architecting Dependable Systems IV, Lecture Notes in Computer Science , Vol. 4615, Lemos, Rogério de; Gacek, Cristina; Romanovsky, Alexander (Eds.), Springer, 2007

[6] B. Boehm, A Spiral Model of Software Development and Enhancement, IEEE Computer, vol. 21, no. 5, May 1988, pp 61-72

[7] T. O. Jackson and J. McDermid, Certification of Neural Networks, ERA Technology Ltd, Report 97-0365, Project 13-01-4745, 1997

[8] B. Littlewood and L. Strigini, Assessment of ultra-high dependability of software-based systems, Communications of the ACM, vol. 36, no. 11, pp. 69–80, November 1993

[9] T. P. Kelly, A Systematic Approach to Safety Case Management, in CAE Methods for Vehicle Crashworthiness and Occupant Safety, and Safety-Critical Systems, Document Number 2004-01-1779, Society of Automotive Engineers, March 2004

[10] U.S. Department of Defense, MilStd 882E, “System Safety Program Requirements" / "Standard Practice for System Safety" Draft 1, February 2006

[11] I. Nabney, Validation of Neural Network Medical Systems, Workshop on Regulatory Issues in Medical Decision Support, October 2001

[12] C. D. Locke, “Software Architecture for Hard Real-Time Applications: Cyclic Executives vs. Fixed Priority Executives”, Real-Time Systems, vol. 4 no. 1, pp37-53, 1992

[13] T. P. Kelly, Arguing Safety – A Systematic Approach to Safety Case Management, DPhil Thesis, YCST-99-05, Department of Computer Science, University of York, 1998

[14] R. A. Weaver, The Safety of Software – Constructing and Assuring Arguments, PhD Thesis, YCST-2004-01, Department of Computer Science, University of York, 2004

[15] Z. Kurd, Artificial Neural Networks in Safety Critical Applications, PhD Thesis, YCST-2006-09, Department of Computer Science, University of York, 2006

[16] D. J. Pumfrey, The Principled Design of Computer System Safety Analyses, DPhil Thesis, Department of Computer Science, University of York

[17] A. Hauge and A. Tonnesen, Use of Artificial Neural Networks in Safety Critical Systems, Faculty of Computer Sciences: Ostfold University College, 2004

[18] B. Cukic, The Need for Verification and Validation Techniques for Adaptive Control System, in Proceedings of the Fifth International Symposium on Autonomous Decentralized Systems (ISADS 2001), pp297–298. IEEE Computer Society, Dallas, Texas, USA, March 2001

[19] J. Voas, Certifying Off-The-Shelf Software Components, IEEE Computer, vol. 31 no. 6, pp53-59, 1998

[20] J. Leonard, H. Durrant-Whyte, Simultaneous map building and localization for an autonomous mobile robot, in Proceedings of the IEEE International Workshop on Intelligent Robots and Systems, pp1442-1447, 1991